ARCHITECTURAL STYLES

part 2

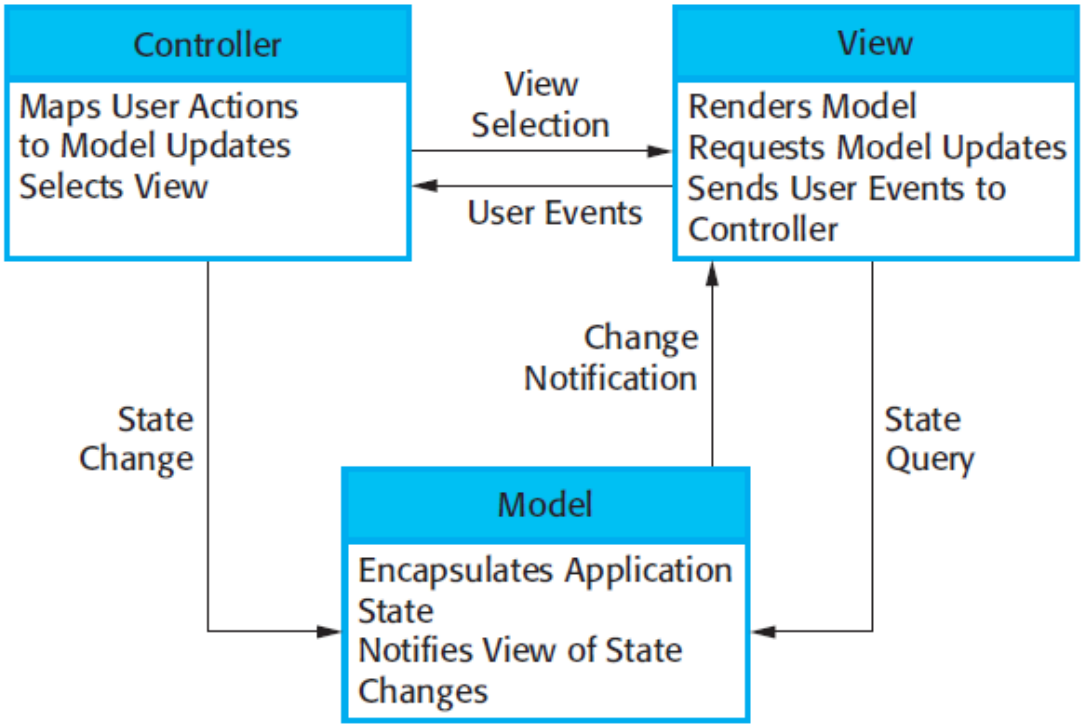
Reflection

- Architectural styles
  - Pipe-and-Filter
  - Layered
  - Client-server
  - Repository/Blackboard
  - Implicit invocation/Message passing
- Today
  - Model-View-Controller
  - Architectural styles in cloud

Model-view controller (MVC) style

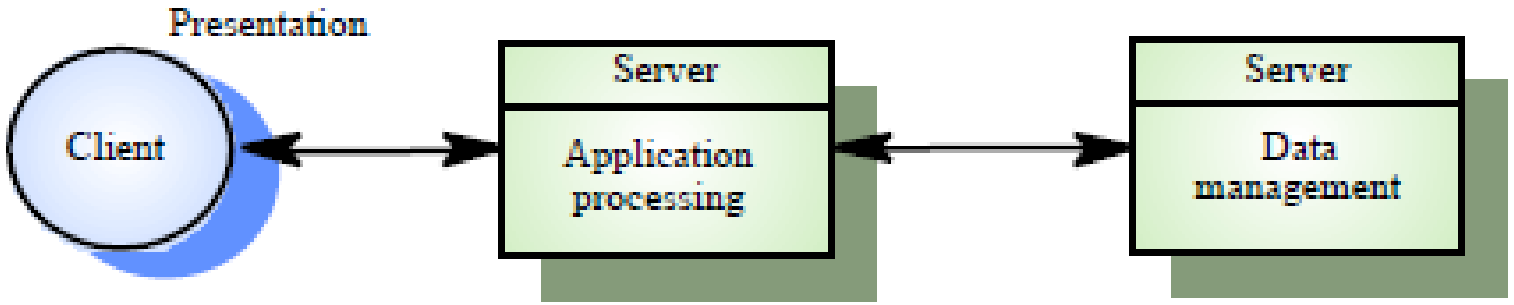
- Enables independence between data, presentation of data and user
- Model component represents knowledge. It manages the behavior and data of the application domain, sends information about its state (to the view), and responds to instructions to change state (usually from the controller)
- View has the duty to manage presentation of information to users
- Controller manages the interaction with the user (e.g. mouse clicks, key pressed, etc.) and informs the model or the view to take appropriate actions

MVC style



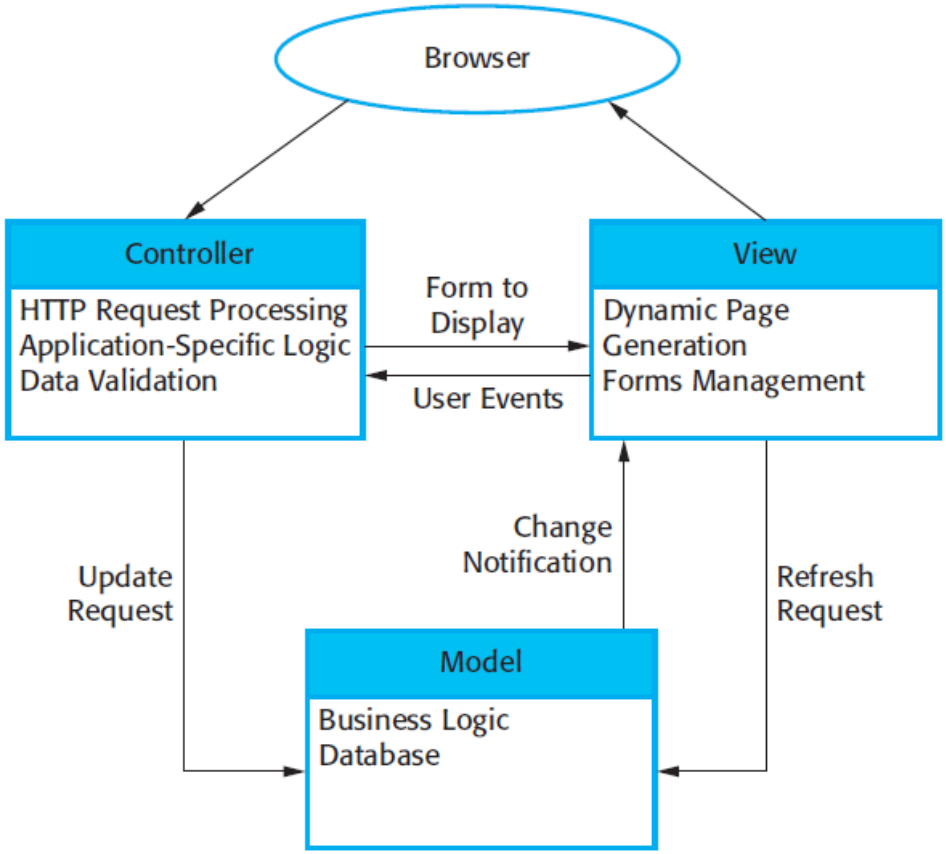
Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Three tier client-server model



- Better performance
- Better security
- …

MVC - example



Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Advantages of MVC

- Great flexibility
  - Easy to maintain and implement future enhancements
  - Clear separation between presentation logic and business logic
  - Easier support for new types of users
- The view is separate and in most systems it undergoes a lot of changes

Disadvantages of MVC

- Even if data model is simple this style may introduce complexity and require a lot of additional code
  - Not suitable for small applications
- Performance issue when frequent updates in the model

ARCHITECTURAL STYLES IN CLOUD

http://msdn.microsoft.com/en-us/library/dn568099.aspx

What is Cloud Computing?

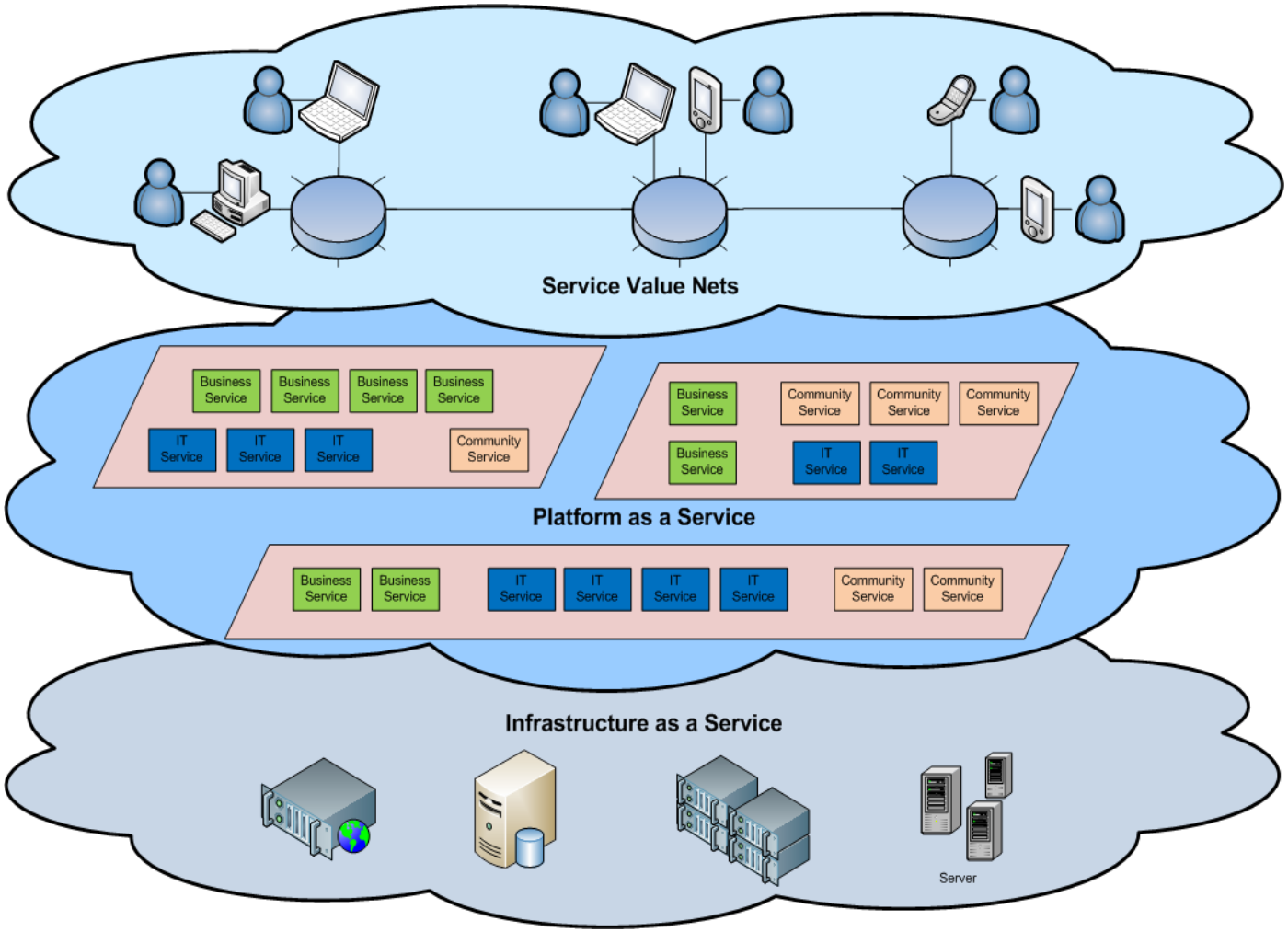
- Cloud Computing is a general term used to describe a new class of network based computing that takes place over the Internet,
  - A group of integrated and networked hardware, software and Internet infrastructure (called a platform).
  - Using the Internet for communication and transport provides hardware, software and networking services to clients
- Platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API (Applications Programming Interface).

What is Cloud Computing?

- Cloud computing is an umbrella term used to refer to Internet based development and services

- A number of characteristics define cloud data, applications services and infrastructure:
  - Remotely hosted: Services or data are hosted on remote infrastructure.
  - Ubiquitous: Services or data are available from anywhere.
  - Commodified: The result is a utility computing model similar to traditional that of traditional utilities, like gas and electricity - you pay for what you would want!

Cloud Flavors?



SaaS
PaaS
IaaS

What is Software as a Service? (SaaS)

- SaaS is a software delivery methodology that provides licensed multi-tenant access to software and its functions remotely as a Web-based service.
  - Usually billed based on usage
  - Usually multi tenant environment
  - Highly scalable architecture
- End users are the consumers of the cloud
  - They use applications (email, calendar, etc.) that happen to be running on a cloud
  - Do not manage the underlying cloud infrastructure

SaaS Examples

facebook

Platform as a Service (PaaS)

- PaaS provides all of the facilities required to support the complete life cycle of building and delivering web applications and services entirely from the Internet.
  - Typically applications must be developed with a particular platform in mind
  - Multi tenant environments
  - Highly scalable multi tier architecture
- Cloud consumers are developers or system administrators
  - Platform provides services like: database, load-balancing, scalability, development environments
  - Consumers deploy applications, using capabilities provided by the cloud provider
  - Consumers does not manage cloud infrastructure, but control the deployed applications
  - Cloud providers offer an Internet-based platform to developers who want to create services but don't want to build their own cloud

PaaS Examples

Infrastructure as a Service (IaaS)

- IaaS is the delivery of technology infrastructure as an on demand scalable service
  - Usually multi tenant virtualized environment
  - Can be coupled with Managed Services for OS and application support
- Cloud consumers are developers or system administrators
  - Processing, storage, networks and other resources, where consumers deploy and run software
  - Consumers have control over operating systems, storage, deployed applications…
  - Cloud users rent storage, computation, and maintenance from cloud providers (pay-as-you-go; like utility)

IaaS Examples

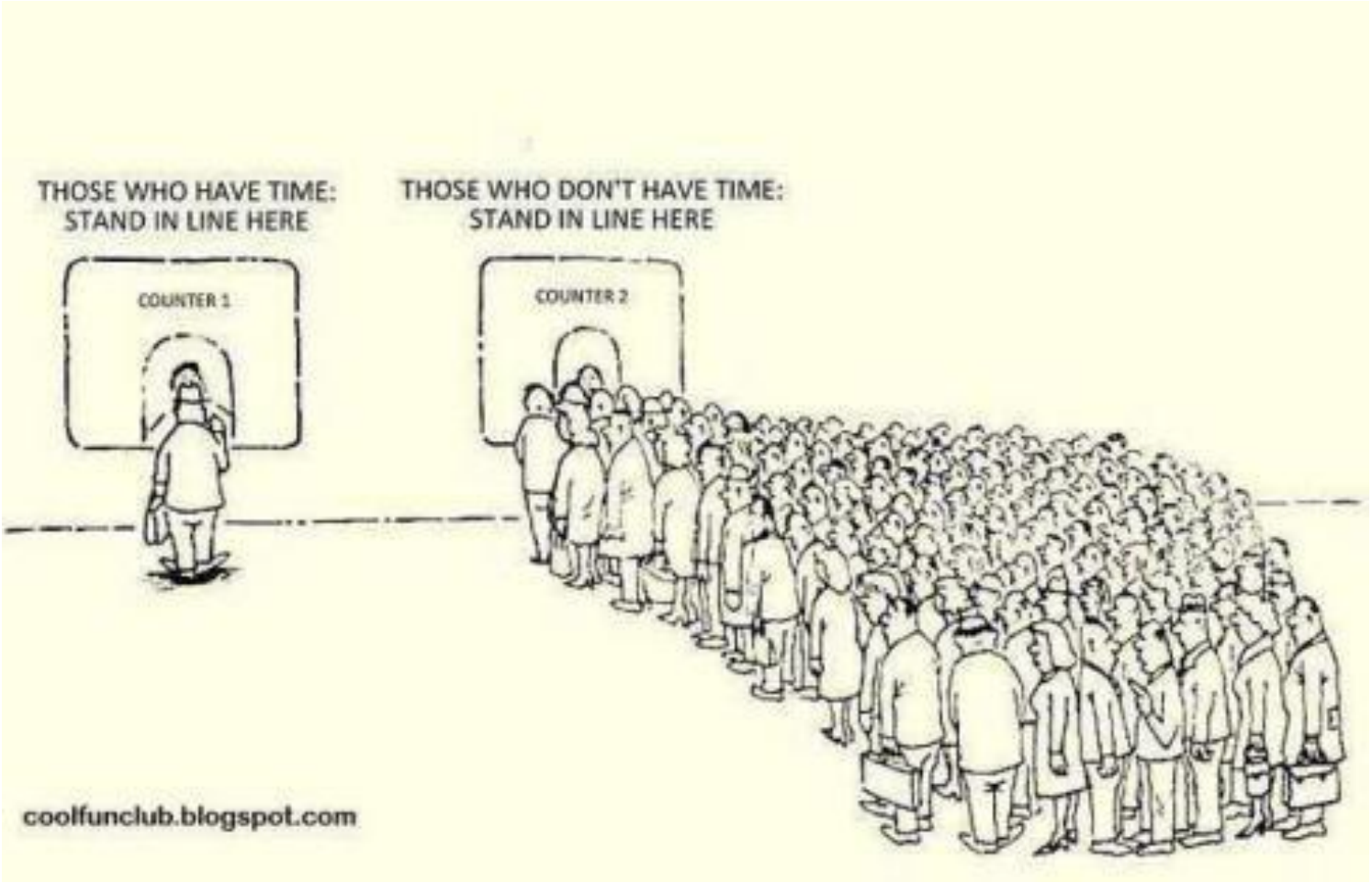
Architectural styles in cloud

- Circuit breaker
- Queue
- Caching
- Sharding

Circuit breaker

- Service fails in a distributed environment
- Classical solution is to implement a timeout for other services that call it
- However this may lead to needless resource consumption in cloud environment
  - Assume hundreds of users, waiting for a failed service and each one waiting for a timeout

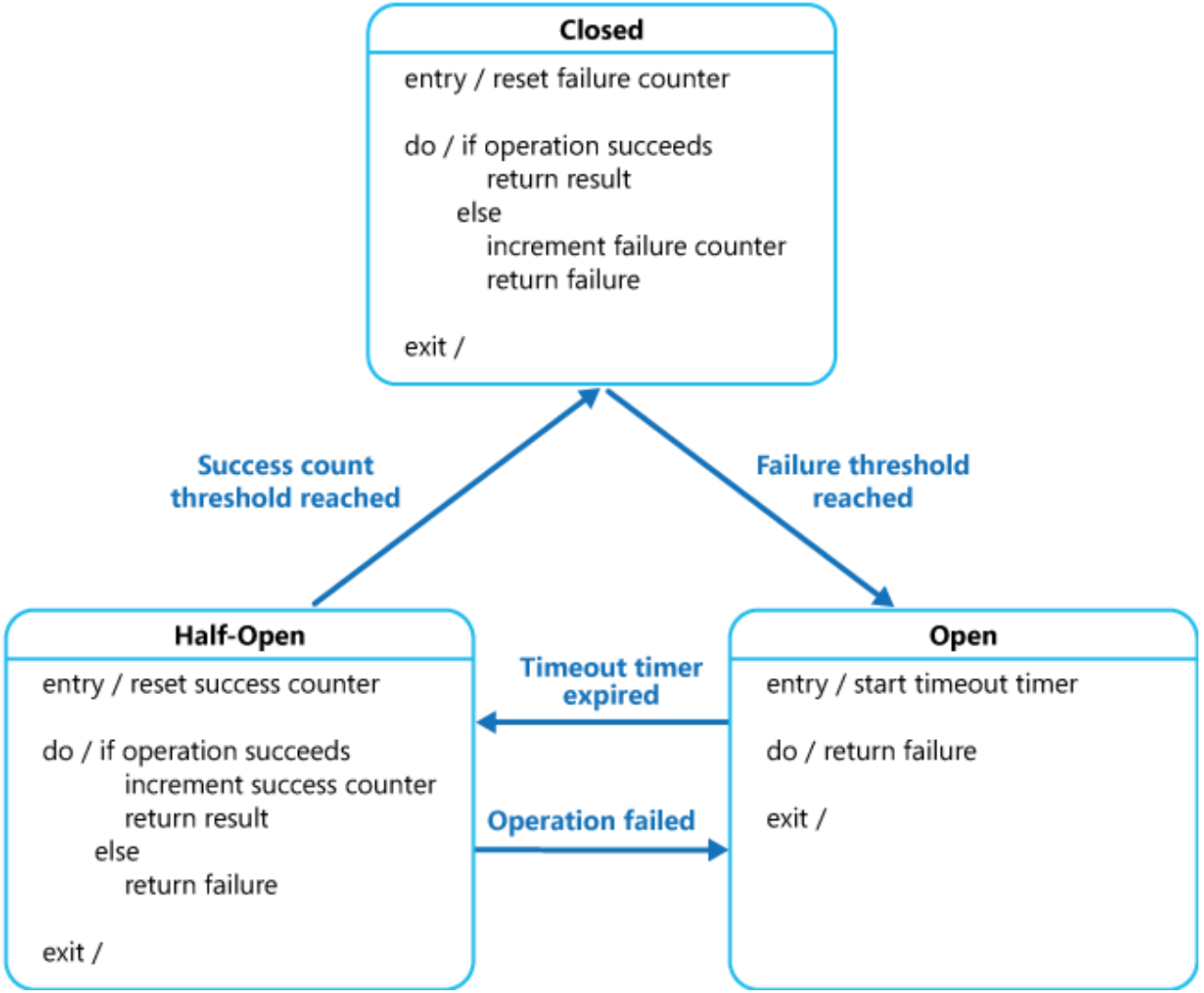
What is the solution?



Circuit breaker pattern

- It prevents an application from performing an operation that is unlikely to succeed
  - E.g. calling a service that is already known to be failed
- Acts as a proxy for operations that may fail
- Monitors the number of recent failures that have occurred, and then use this information to decide whether to allow the operation to proceed, or simply return an exception immediately

Circuit Breaker Pattern



Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

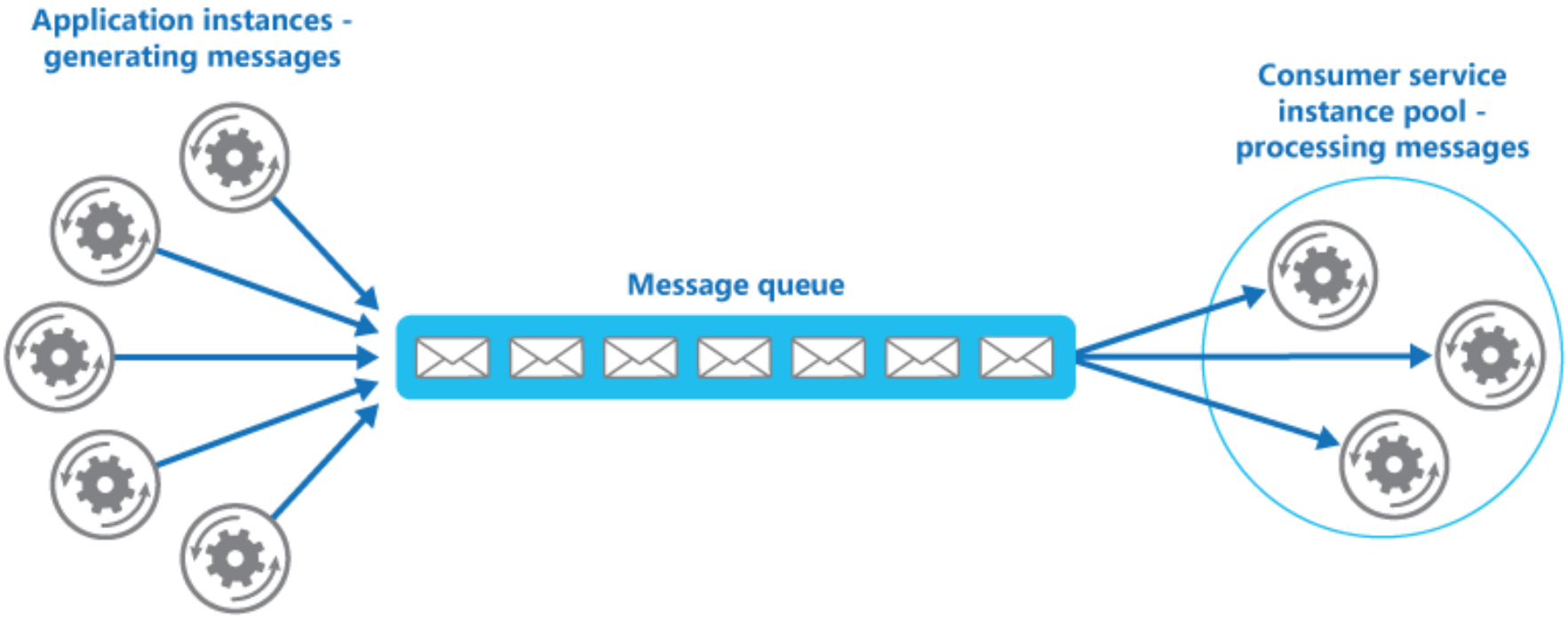
Circuit Breaker Pattern

- Helps to increase system dependability
  - A requesting application or large number of applications would not unnecessarily wait for timeout
    - When the service is known to be down
    - The network connection is temporarily down
    - The service is known to be very busy and not capable to respond

Queue

- Problem definition
  - In cloud you may have services that are flooded by a large number of concurrent requests by other services. In this case they may be overloaded or experience peak loads. You should find a solution to smooth heavy loads that may cause the service to fail or the calling task to time out.
- Different kinds of queues
  - Standard queue
  - Priority queue
  - Fixed length queue

Standard queue

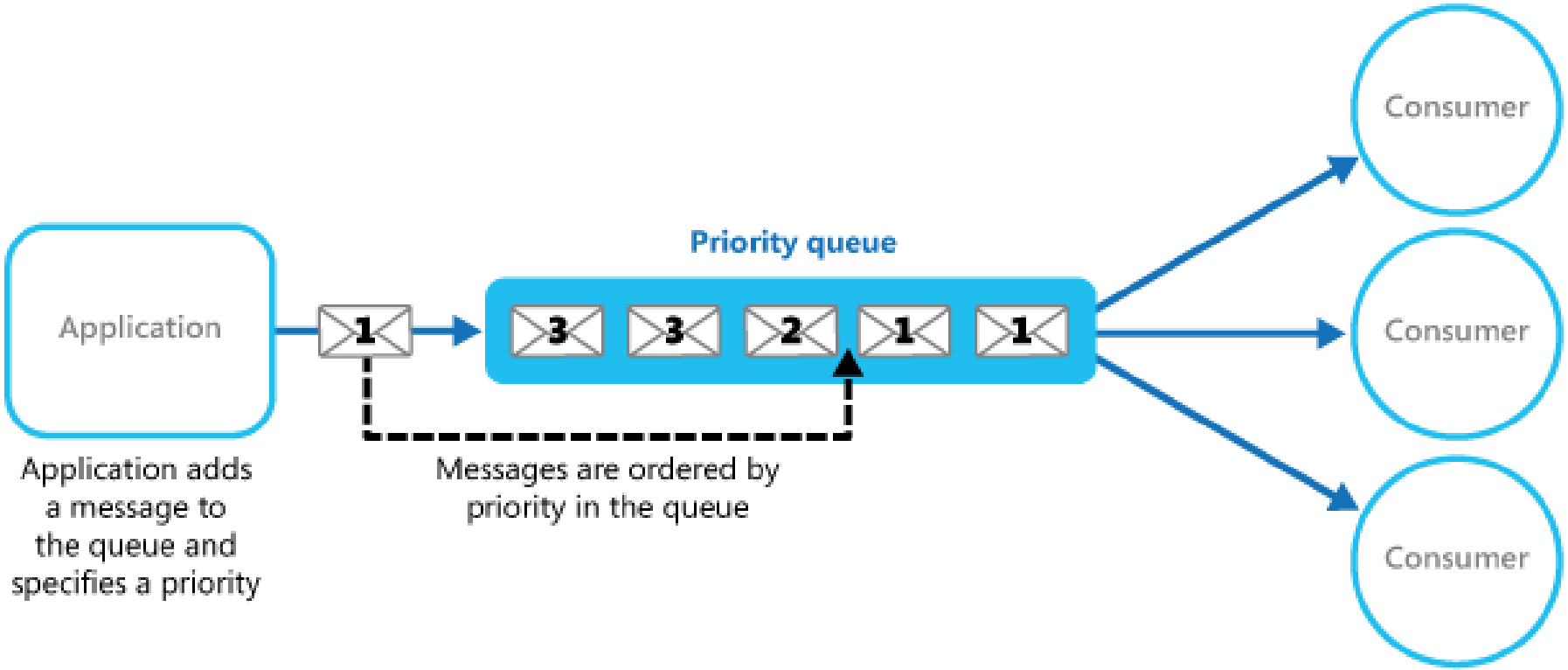


Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Standard queue

- The queue acts as a buffer, storing the messages until they are retrieved by the service.
- The service retrieves the messages from the queue and processes them
- Minimizes availability risks by a large number of concurrent requests.

Priority queue



Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Priority queue

- Implements a policy to sort incoming request according to their priority
- Priority of request may be set by the sending applications or by the queue itself

Fixed length queue

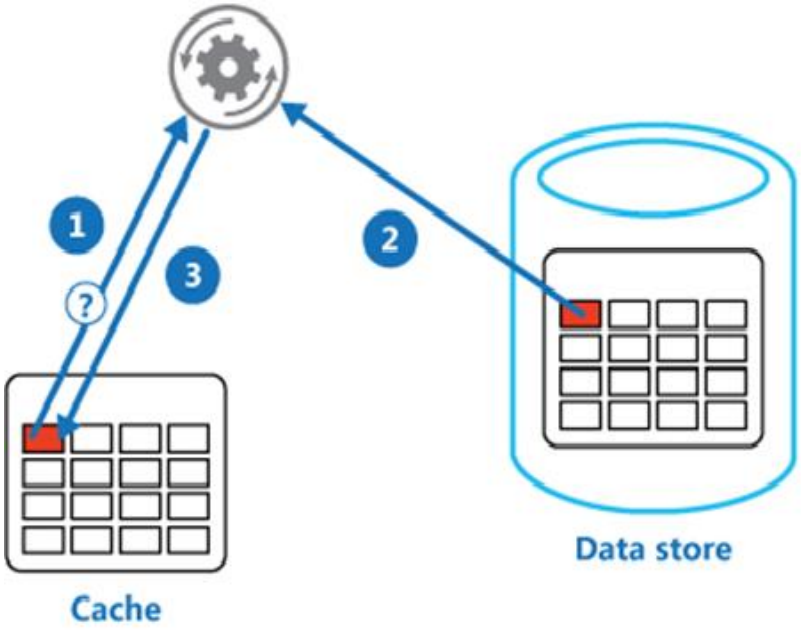
- What about denial of service attacks?

- We may design the queue in order to send an exception when a specific (extraordinary high for the service) amount of messages is reached
- In this case the requesting task will know that its request would not be processed and may take appropriate action, without waiting for the timeout

Cache

- Used to optimize repeated access to information held in a data store
- Will cached data be always completely consistent with the data in the data store?

Cache

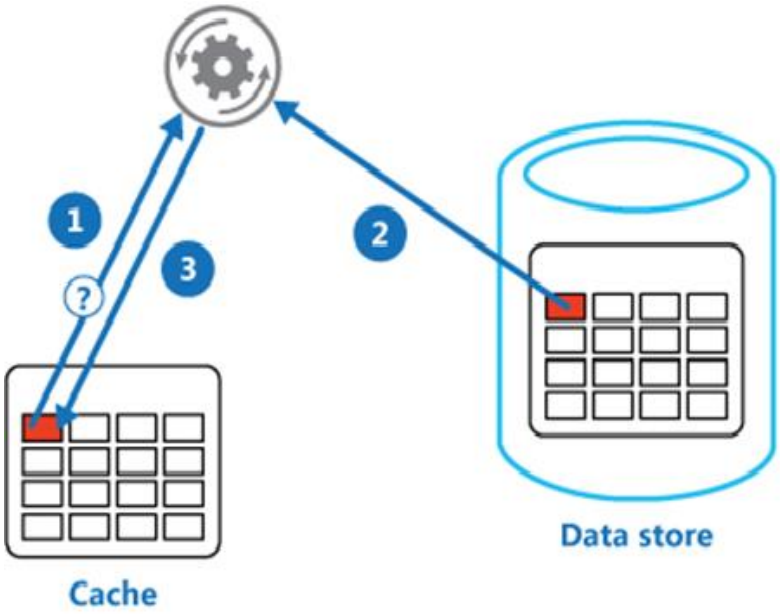


- Reading
  1. Look for item in the cache
  2. If it is not there, retrieve it from the cache
  3. Store a copy into the cache
- Writing
  1. Make the modification to the data store
  2. Invalidate the corresponding item in the cache.

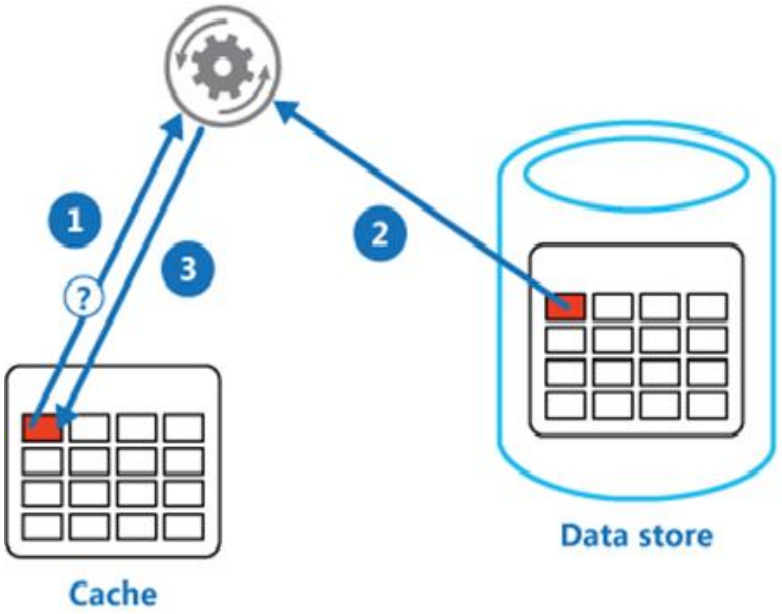
Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Facebook issue

Master



Slave



How is synchronization between two data stores managed

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Sharding style

- The aim is to increase scalability when using big amounts of data
- The idea is to divide a data store into a set of horizontal partitions, called shards
- Motivation is to increase:
  - Storage space
  - Computing resources
  - Network bandwidth
  - Geographic disperse

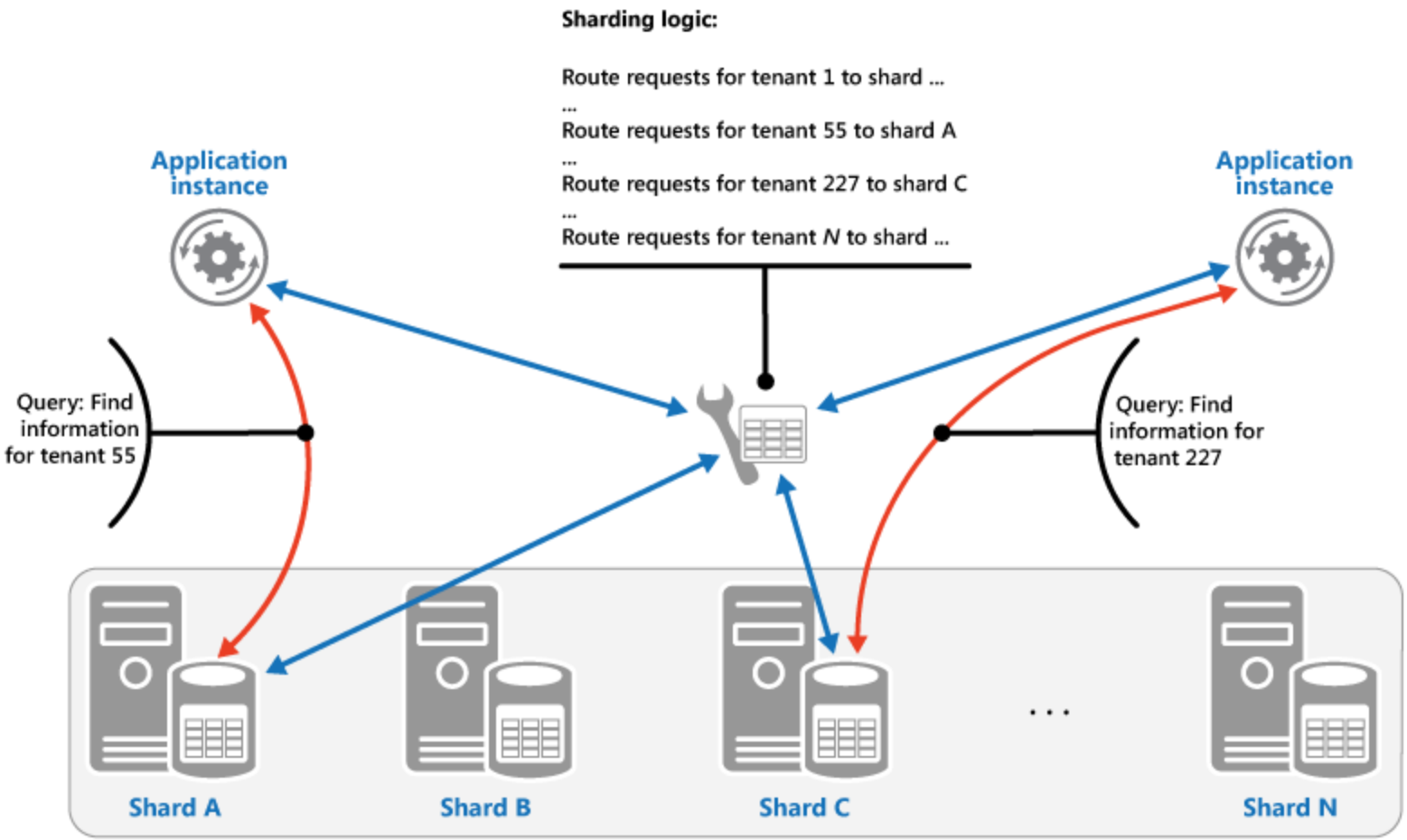
Sharding style

- Decision about how to divide data between shards is important
  - A shard typically contains items that fall within a specified range determined by one or more attributes of the data.
  - Data attributes form the *shard key* (aka *partition key*).
  - Shard key should be static. It should not be based on data that might change.
- Sharding physically organizes the data - when an application stores and retrieves data, the sharding logic directs the application to the appropriate shard
- Implementation issues about sharding logic
  - May be implemented as part of the data access code in the application
  - May be implemented by the data storage system if it transparently supports sharding.

Sharding style

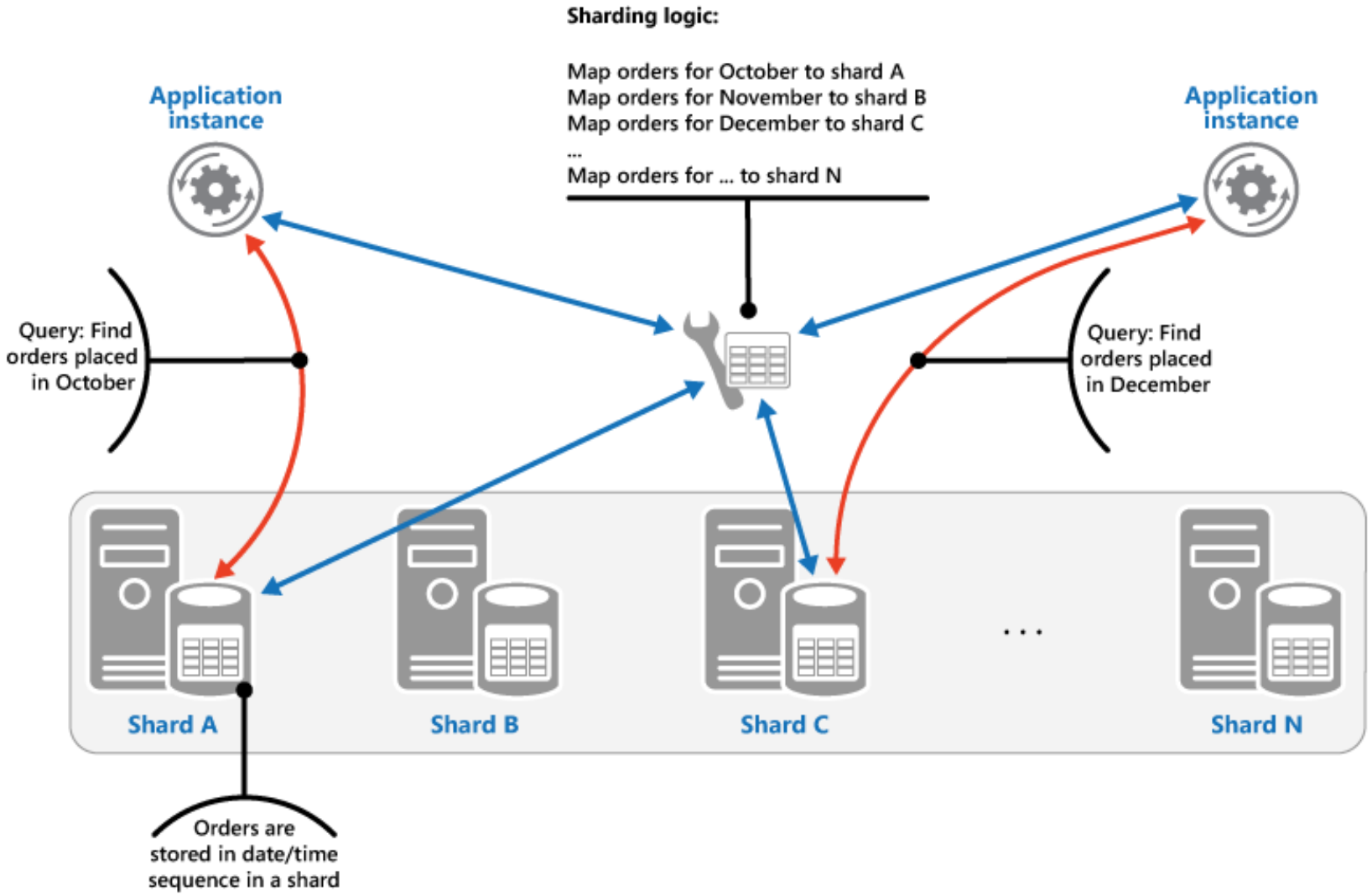
- Three basic strategies exist for implementation of sharding logic
  - Lookup strategy
  - Range strategy
  - Hash strategy

Lookup sharding strategy



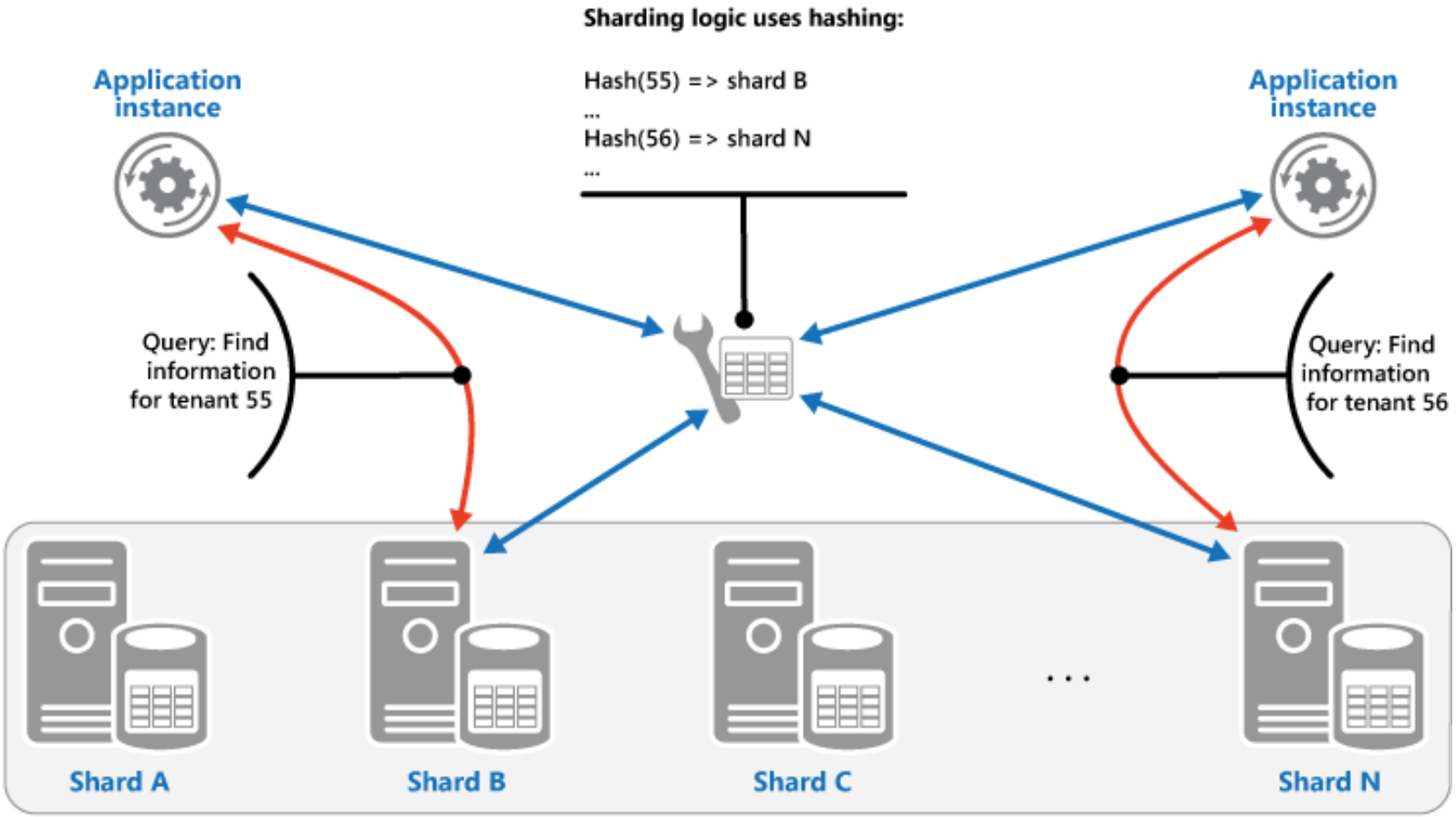
Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Range sharding strategy



Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Hash sharding strategy



Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Sharding advantages

- Better data management - abstraction of data physical location provides
  - Control over which shards contain which data
- Increased performance of the data storage

Sharding disadvantages

- Application performance issues
  - Overhead when determining the location of each data
  - More performance overhead when data that matches a single request is distribute among many shards
- Shards may contain misbalanced amount of data
- Sometimes it is extremly difficult to design a shard key that matches the requirements of every possible query against the data.